Datamining Relational Databases for Regression Analysis

#235

Paul Harten[1], Henry Helgen[2], Wilson Melendez[2]

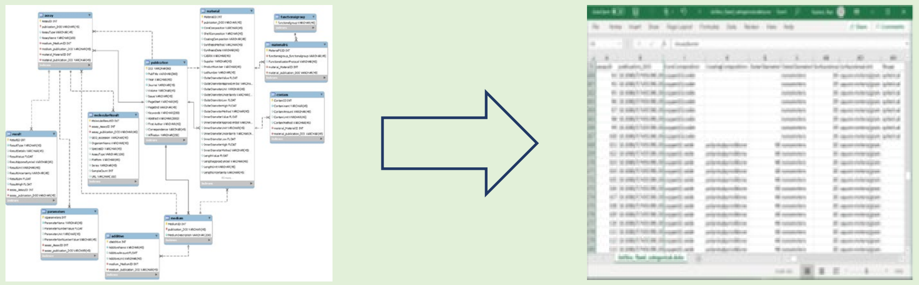
[1]US EPA/ORD, Center for Computational Toxicology and Exposure; [2]General Dynamics Information Technology, Contractor to US EPA

OBJECTIVES

- Datamine EPA's database NaKnowBase that contains details about thousands of experiments involving nanomaterials conducted by ORD scientists.
- Predict the adverse effects of nanomaterials from models built from a rich number of examples via regression analysis.

APPROACH

Flatten the relational database NaKnowBase, pivoting on tables to place multiple entries for each experiment into one unique row, with all experiments placed in a spreadsheet format.



MAIN RESULTS

- EPA's relational database NaKnowBase is flattened into a spreadsheet format. Other organizations can copy and modify the nanomaterial data for their own analysis.
- Initial steps for EPA's Chemical Safety for Sustainability (CSS), Emerging Materials and Technology (EMT) CSS 3.2.3 Predictive model - nanoQSAR

IMPACT

- Approaches to develop nanoQSAR models from relational databases will facilitate the prediction of adverse activities of novel nanomaterials.
- EPA Office of Research and Development can use nanoQSAR models for additional investigative methods.
- **For more information, contact:** Paul Harten, harten.paul@epa.gov

This work does not reflect EPA policy.

Datamining Relational Databases for Regression Analysis

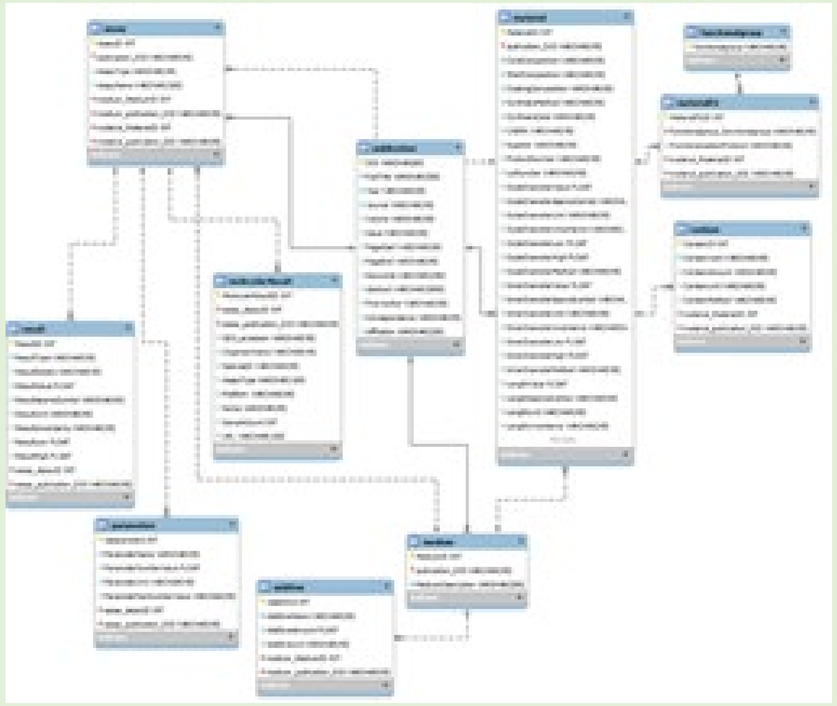
Overview

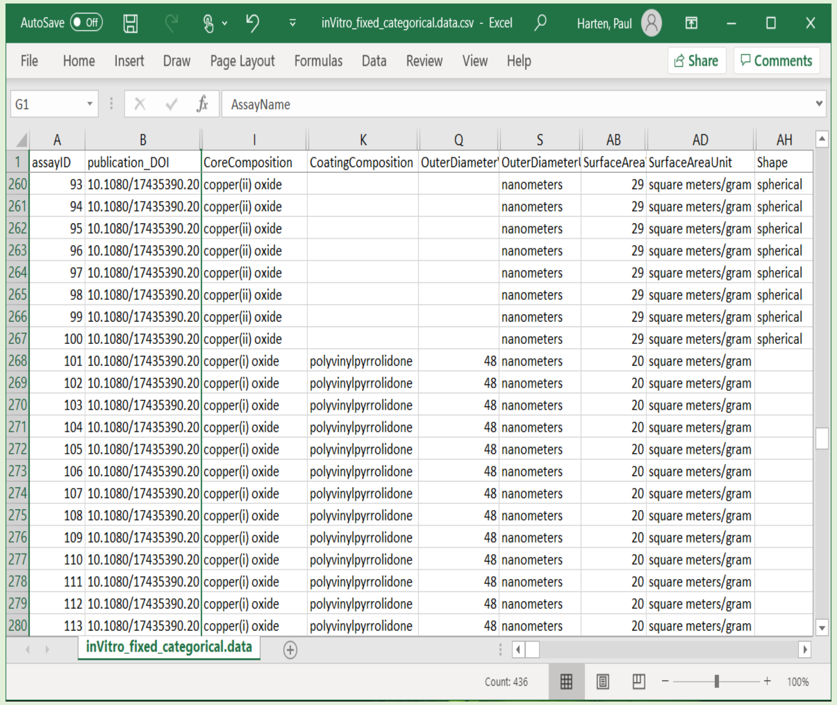
- Most regression analysis algorithms have training and testing samples given in spreadsheet format, a sample for each row.
- However, sometimes the parameters, variables, and results of experiments are held in relational databases to reduce the space needed to itemize corresponding features.
- Because of the structure of relational databases, datamining examples of detailed experiments into a spreadsheet format can be particularly difficult.
- This method translates your relational database into a spreadsheet format CSV file ready for regression analysis.

Datamining Relational Databases for Regression Analysis

APPROACH

Flatten the relational database NaKnowBase, pivoting on tables to place multiple entries for each experiment into one unique row, with all experiments placed in a spreadsheet format.



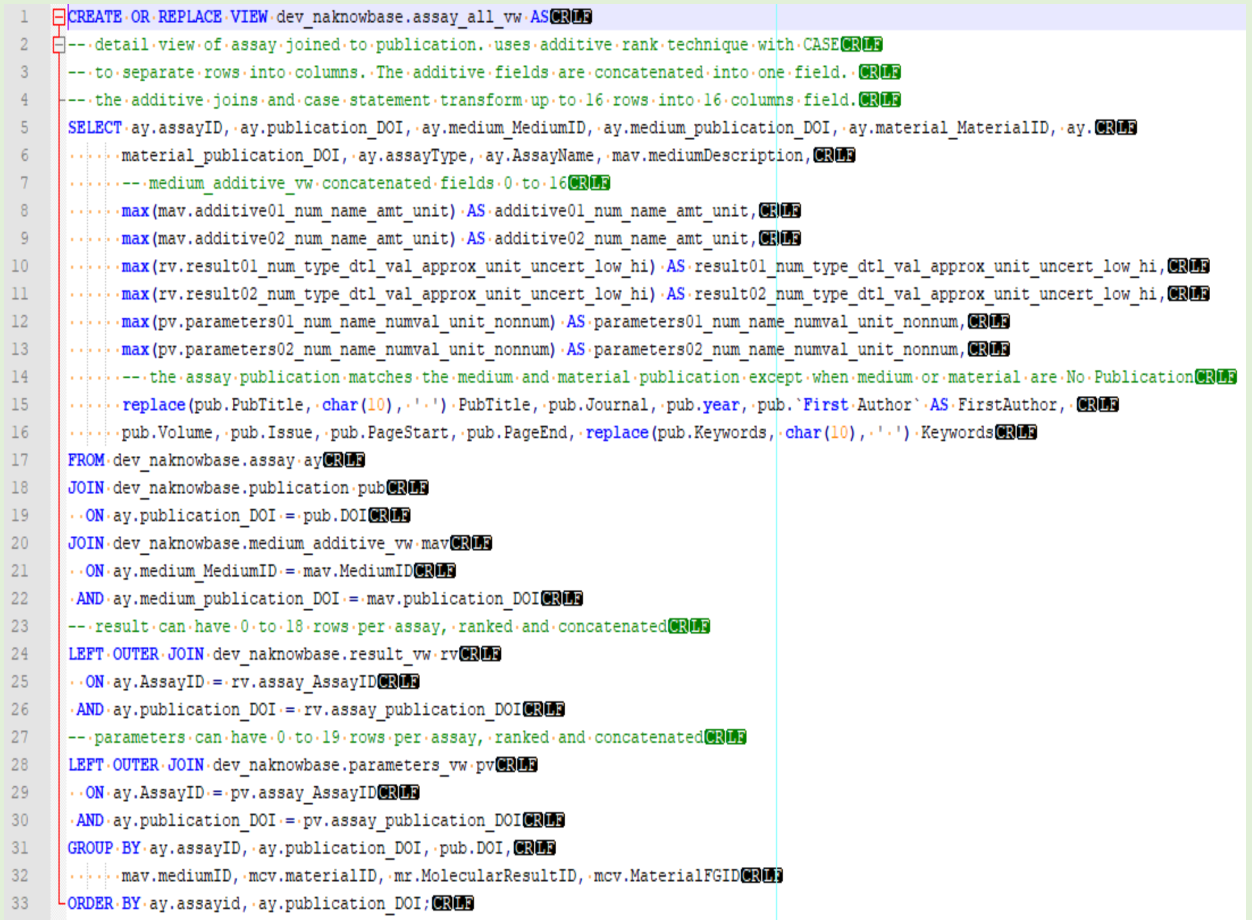


	A	B	I	K	Q	S	AB	AD	AH
1	assayID	publication_DOI	CoreComposition	CoatingComposition	OuterDiameter'	OuterDiameter(	SurfaceArea	SurfaceAreaUnit	Shape
260	93	10.1080/17435390.20	copper(ii) oxide			nanometers	29	square meters/gram	spherical
261	94	10.1080/17435390.20	copper(ii) oxide			nanometers	29	square meters/gram	spherical
262	95	10.1080/17435390.20	copper(ii) oxide			nanometers	29	square meters/gram	spherical
263	96	10.1080/17435390.20	copper(ii) oxide			nanometers	29	square meters/gram	spherical
264	97	10.1080/17435390.20	copper(ii) oxide			nanometers	29	square meters/gram	spherical
265	98	10.1080/17435390.20	copper(ii) oxide			nanometers	29	square meters/gram	spherical
266	99	10.1080/17435390.20	copper(ii) oxide			nanometers	29	square meters/gram	spherical
267	100	10.1080/17435390.20	copper(ii) oxide			nanometers	29	square meters/gram	spherical
268	101	10.1080/17435390.20	copper(i) oxide	polyvinylpyrrolidone	48	nanometers	20	square meters/gram	
269	102	10.1080/17435390.20	copper(i) oxide	polyvinylpyrrolidone	48	nanometers	20	square meters/gram	
270	103	10.1080/17435390.20	copper(i) oxide	polyvinylpyrrolidone	48	nanometers	20	square meters/gram	
271	104	10.1080/17435390.20	copper(i) oxide	polyvinylpyrrolidone	48	nanometers	20	square meters/gram	
272	105	10.1080/17435390.20	copper(i) oxide	polyvinylpyrrolidone	48	nanometers	20	square meters/gram	
273	106	10.1080/17435390.20	copper(i) oxide	polyvinylpyrrolidone	48	nanometers	20	square meters/gram	
274	107	10.1080/17435390.20	copper(i) oxide	polyvinylpyrrolidone	48	nanometers	20	square meters/gram	
275	108	10.1080/17435390.20	copper(i) oxide	polyvinylpyrrolidone	48	nanometers	20	square meters/gram	
276	109	10.1080/17435390.20	copper(i) oxide	polyvinylpyrrolidone	48	nanometers	20	square meters/gram	
277	110	10.1080/17435390.20	copper(i) oxide	polyvinylpyrrolidone	48	nanometers	20	square meters/gram	
278	111	10.1080/17435390.20	copper(i) oxide	polyvinylpyrrolidone	48	nanometers	20	square meters/gram	
279	112	10.1080/17435390.20	copper(i) oxide	polyvinylpyrrolidone	48	nanometers	20	square meters/gram	
280	113	10.1080/17435390.20	copper(i) oxide	polyvinylpyrrolidone	48	nanometers	20	square meters/gram	

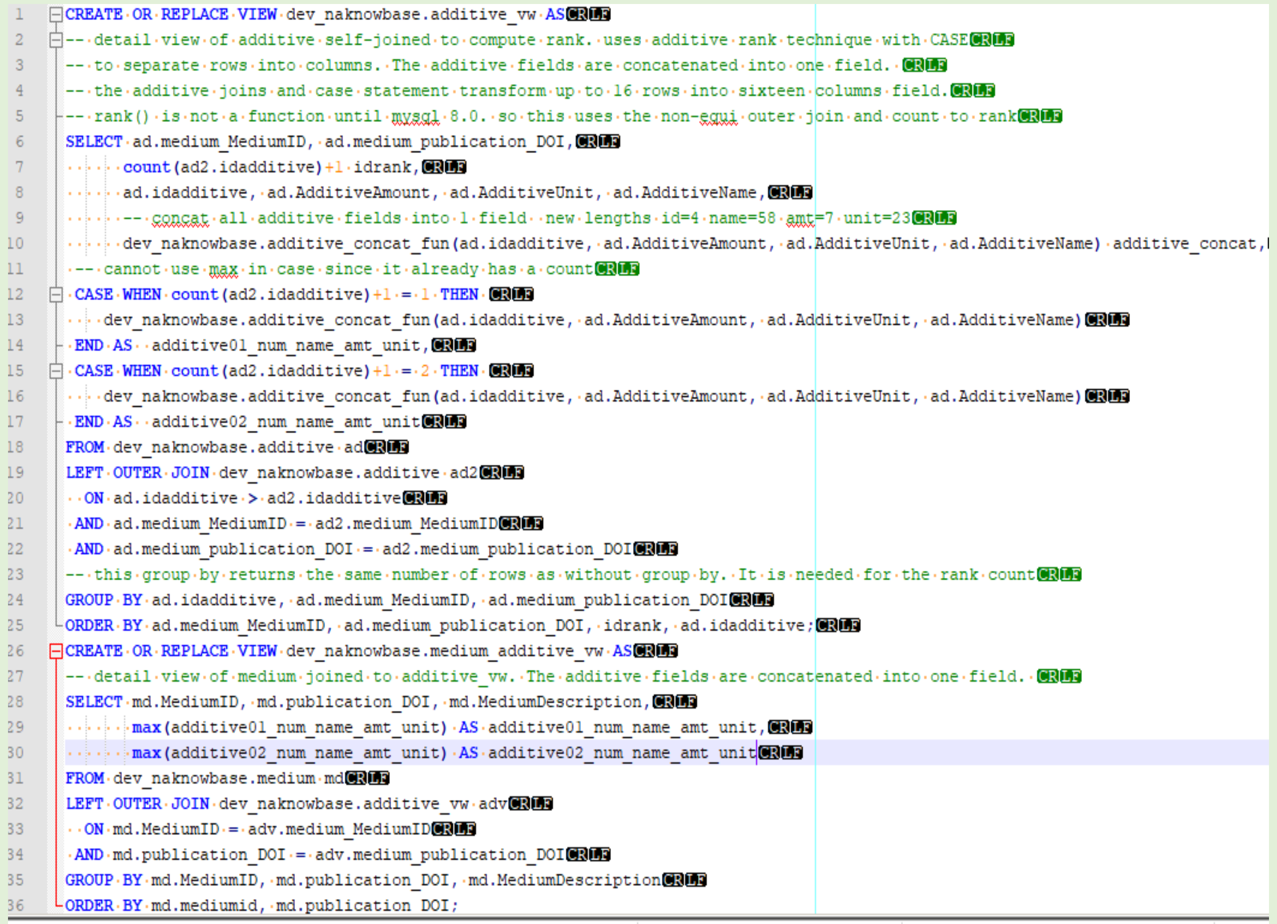
Datamining Relational Databases for Regression Analysis

APPROACH (cont.)

MySQL script to flatten the relational database NaKnowBase, pivoting on tables to place multiple concatenated entries for each experiment into one unique row:



```sql
CREATE OR REPLACE VIEW dev_naknowbase.assay_all_vw AS
-- detail view of assay joined to publication. uses additive rank technique with CASE
-- to separate rows into columns. The additive fields are concatenated into one field. 
-- the additive joins and case statement transform up to 16 rows into 16 columns field.
SELECT ay.assayID, ay.publication_DOI, ay.medium_MediumID, ay.medium_publication_DOI, ay.material_MaterialID, ay.
     material_publication_DOI, ay.assayType, ay.AssayName, mav.mediumDescription,
     -- medium_additive_vw concatenated fields 0 to 16
     max(mav.additive01_num_name_amt_unit) AS additive01_num_name_amt_unit,
     max(mav.additive02_num_name_amt_unit) AS additive02_num_name_amt_unit,
     max(rv.result01_num_type_dtl_val_approx_unit_uncert_low_hi) AS result01_num_type_dtl_val_approx_unit_uncert_low_hi,
     max(rv.result02_num_type_dtl_val_approx_unit_uncert_low_hi) AS result02_num_type_dtl_val_approx_unit_uncert_low_hi,
     max(pv.parameters01_num_name_numval_unit_nonnum) AS parameters01_num_name_numval_unit_nonnum,
     max(pv.parameters02_num_name_numval_unit_nonnum) AS parameters02_num_name_numval_unit_nonnum,
     -- the assay publication matches the medium and material publication except when medium or material are No Publication
     replace(pub.PubTitle, char(10), ' ') PubTitle, pub.Journal, pub.year, pub.`First Author` AS FirstAuthor, 
     pub.Volume, pub.Issue, pub.PageStart, pub.PageEnd, replace(pub.Keywords, char(10), ' ') Keywords
FROM dev_naknowbase.assay ay
JOIN dev_naknowbase.publication pub
  ON ay.publication_DOI = pub.DOI
JOIN dev_naknowbase.medium_additive_vw mav
  ON ay.medium_MediumID = mav.MediumID
 AND ay.medium_publication_DOI = mav.publication_DOI
-- result can have 0 to 18 rows per assay, ranked and concatenated
LEFT OUTER JOIN dev_naknowbase.result_vw rv
  ON ay.AssayID = rv.assay_AssayID
 AND ay.publication_DOI = rv.assay_publication_DOI
-- parameters can have 0 to 19 rows per assay, ranked and concatenated
LEFT OUTER JOIN dev_naknowbase.parameters_vw pv
  ON ay.AssayID = pv.assay_AssayID
 AND ay.publication_DOI = pv.assay_publication_DOI
GROUP BY ay.assayID, ay.publication_DOI, pub.DOI,
     mav.mediumID, mcv.materialID, mr.MolecularResultID, mcv.MaterialFGID
ORDER BY ay.assayid, ay.publication_DOI;
```

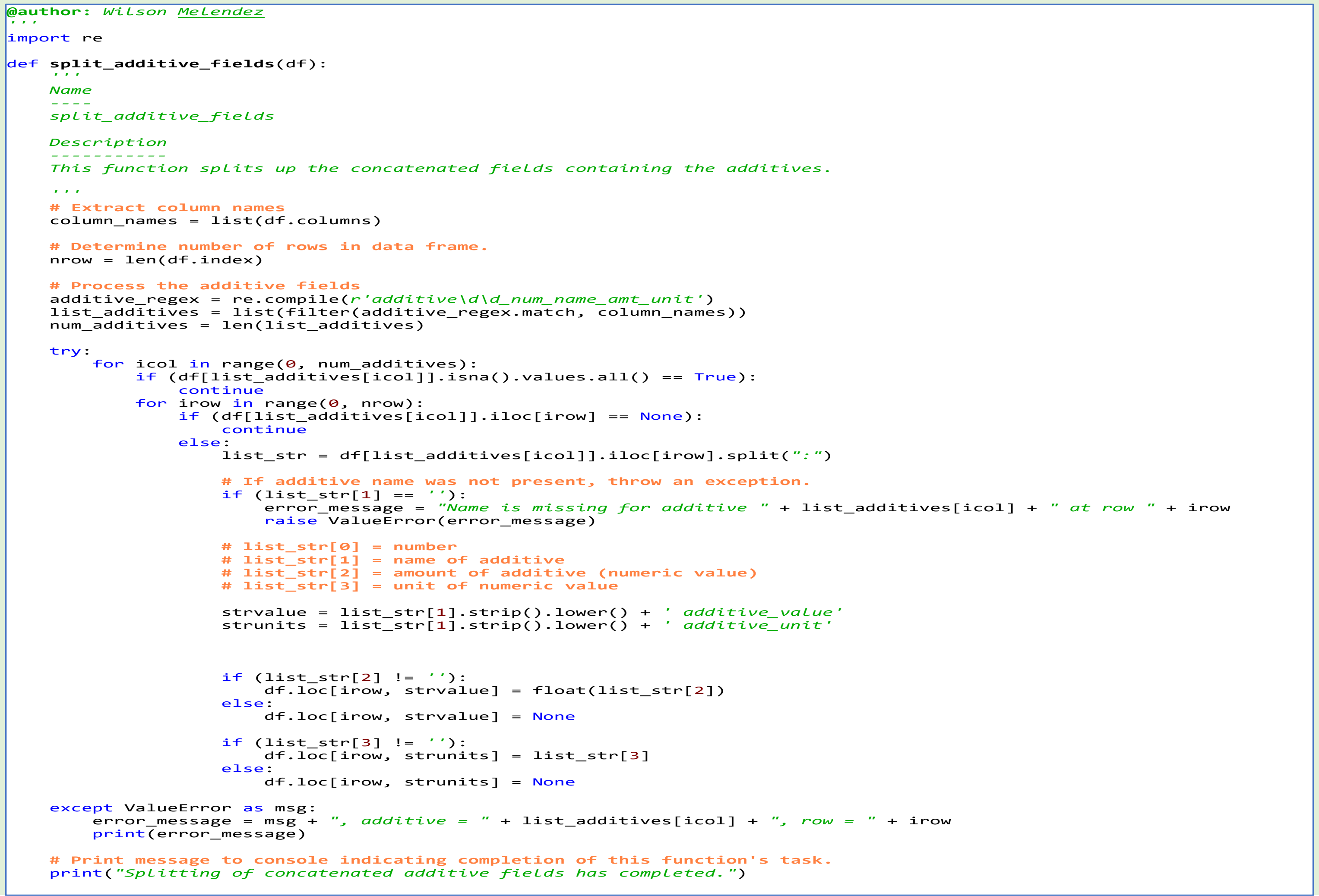


```sql
CREATE OR REPLACE VIEW dev_naknowbase.additive_vw AS
-- detail view of additive self-joined to compute rank. uses additive rank technique with CASE
-- to separate rows into columns. The additive fields are concatenated into one field. 
-- the additive joins and case statement transform up to 16 rows into sixteen columns field.
-- rank() is not a function until mysql 8.0. so this uses the non-equi outer join and count to rank
SELECT ad.medium_MediumID, ad.medium_publication_DOI,
     count(ad2.idadditive)+1 idrank,
     ad.idadditive, ad.AdditiveAmount, ad.AdditiveUnit, ad.AdditiveName,
     -- concat all additive fields into 1 field  new lengths id=4 name=58 amt=7 unit=23
     dev_naknowbase.additive_concat_fun(ad.idadditive, ad.AdditiveAmount, ad.AdditiveUnit, ad.AdditiveName) additive_concat,
 -- cannot use max in case since it already has a count
 CASE WHEN count(ad2.idadditive)+1 = 1 THEN 
   dev_naknowbase.additive_concat_fun(ad.idadditive, ad.AdditiveAmount, ad.AdditiveUnit, ad.AdditiveName)
 END AS  additive01_num_name_amt_unit,
 CASE WHEN count(ad2.idadditive)+1 = 2 THEN 
   dev_naknowbase.additive_concat_fun(ad.idadditive, ad.AdditiveAmount, ad.AdditiveUnit, ad.AdditiveName)
 END AS  additive02_num_name_amt_unit
FROM dev_naknowbase.additive ad
LEFT OUTER JOIN dev_naknowbase.additive ad2
  ON ad.idadditive > ad2.idadditive
 AND ad.medium_MediumID = ad2.medium_MediumID
 AND ad.medium_publication_DOI = ad2.medium_publication_DOI
-- this group by returns the same number of rows as without group by. It is needed for the rank count
GROUP BY ad.idadditive, ad.medium_MediumID, ad.medium_publication_DOI
ORDER BY ad.medium_MediumID, ad.medium_publication_DOI, idrank, ad.idadditive;
CREATE OR REPLACE VIEW dev_naknowbase.medium_additive_vw AS
-- detail view of medium joined to additive_vw. The additive fields are concatenated into one field. 
SELECT md.MediumID, md.publication_DOI, md.MediumDescription,
      max(additive01_num_name_amt_unit) AS additive01_num_name_amt_unit,
      max(additive02_num_name_amt_unit) AS additive02_num_name_amt_unit
FROM dev_naknowbase.medium md
LEFT OUTER JOIN dev_naknowbase.additive_vw adv
  ON md.MediumID = adv.medium_MediumID
 AND md.publication_DOI = adv.medium_publication_DOI
GROUP BY md.MediumID, md.publication_DOI, md.MediumDescription
ORDER BY md.mediumid, md.publication_DOI;
```

Datamining Relational Databases for Regression Analysis

APPROACH (cont.)

Python code to interpret the multiple concatenated entries for each row into multiple columns:



```python
@author: Wilson Melendez
'''
import re

def split_additive_fields(df):
    '''
    Name
    ----
    split_additive_fields

    Description
    -----------
    This function splits up the concatenated fields containing the additives.

    '''
    # Extract column names
    column_names = list(df.columns)

    # Determine number of rows in data frame.
    nrow = len(df.index)

    # Process the additive fields
    additive_regex = re.compile(r'additive\d\d_num_name_amt_unit')
    list_additives = list(filter(additive_regex.match, column_names))
    num_additives = len(list_additives)

    try:
        for icol in range(0, num_additives):
            if (df[list_additives[icol]].isna().values.all() == True):
                continue
            for irow in range(0, nrow):
                if (df[list_additives[icol]].iloc[irow] == None):
                    continue
                else:
                    list_str = df[list_additives[icol]].iloc[irow].split(":")

                    # If additive name was not present, throw an exception.
                    if (list_str[1] == ''):
                        error_message = "Name is missing for additive " + list_additives[icol] + " at row " + irow
                        raise ValueError(error_message)

                    # list_str[0] = number
                    # list_str[1] = name of additive
                    # list_str[2] = amount of additive (numeric value)
                    # list_str[3] = unit of numeric value

                    strvalue = list_str[1].strip().lower() + ' additive_value'
                    strunits = list_str[1].strip().lower() + ' additive_unit'


                    if (list_str[2] != ''):
                        df.loc[irow, strvalue] = float(list_str[2])
                    else:
                        df.loc[irow, strvalue] = None

                    if (list_str[3] != ''):
                        df.loc[irow, strunits] = list_str[3]
                    else:
                        df.loc[irow, strunits] = None

    except ValueError as msg:
        error_message = msg + ", additive = " + list_additives[icol] + ", row = " + irow
        print(error_message)

    # Print message to console indicating completion of this function's task.
    print("Splitting of concatenated additive fields has completed.")
```